Tejas Nitin Joshi
#102181822

tejas.joshi@colorado.edu

ELEC 5623 RTES

LAB 1 REPORT

Familiarization with Rate Monotonic and Deadline Monotonic policy and feasibility tests, understanding of sufficient vs. necessary and sufficient feasibility

1. posix_rt_timers.c with 10ms resolution

This code relies heavily on the posix_rt_timers.c implementation. So a little explanation on important portions of posix_rt_timers.c and the important functions involved in crating timers.

Let us start with the entry point function here, i.e. ptimer_test(). Here I came across new structures and completely new timer initialization code , which seem very convoluted at the first glance.

Let's begin!

Setting up a timer requires the use of mainly three functions, with arguments provided in proper type format

This would require the functions

1. timer_create()
2. timer_settime()
3. All the satisfying arguments for the above to functions with proper time resolution settings.

timer_create()

This function basically does the pre-configuration work for the timer, It requires the following arguments , first is the clock ID , we are using CLOCK_REALTIME for this.

The second argument is a pointer to a structure "sigevent" designed to link a timer expiration event creation, more on this after the final argument and finally a pointer to a timer_t type user assigned variable to store the unique timer ID generated.

The sigevent is a structure with members which are defined as follows:

```
union sigval {          /* Data passed with notification */
           int     sival_int;         /* Integer value */
           void   *sival_ptr;         /* Pointer value */
       };
struct sigevent {
           int          sigev_notify; /* Notification method */
           int          sigev_signo;  /* Notification signal */
           union sigval sigev_value;  /* Data passed with notification */
           void       (*sigev_notify_function) (union sigval);
                            /* Function used for thread
                               notification (SIGEV_THREAD) */
           void        *sigev_notify_attributes;
                            /* Attributes for notification thread
                               (SIGEV_THREAD) */
           pid_t        sigev_notify_thread_id;
                            /* ID of thread to signal (SIGEV_THREAD_ID) */
       };
```

The sigev_signo specifies the signal number , POSIX allows the user 32 signals with numbers starting from 32 to 64 and are denoted as SIGRTMIN(i.e.32) and SIGRTMAX(i.e. 64), The notation of specifying the signal is always in the form SIGRTMIN+n, where n is the offset from 32 , We use SIGRTMIN+1 (i.e signal 33) for the implementation.

timer_settime()

int timer_settime(

timer_t timerid, int flags,

const struct itimerspec *new_value,

struct itimerspec * old_value);

This function is responsible for armin/disarming the timer with the right conditions, the first argument requires a timer_ID value to identify the timer which we obtained from timer_create() function and then requires a flag which can be set to 0 for a timing mesure that runsbased on the current system time or can be specified with a flag ABS_TIMER which means it takes the e value loaded in itimerspec structure , sub member ->it_value as absolute reference,

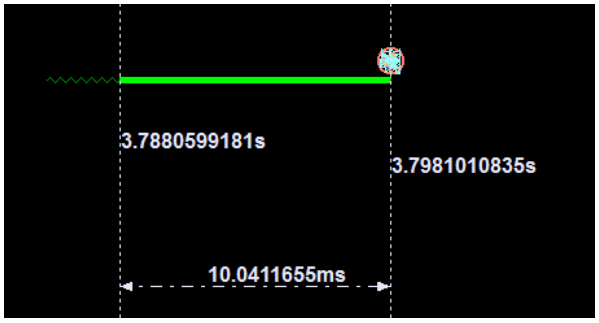
The third argument nrefrences to the timer count, which is a pointer to the a itimerspec struct, which has two sub structures of type timespec of type it_value a.k.a interval value and the it_interval also referred to as timer period. These two structures are of the same type timerspec which specify time in two field combination tv_sec and tv_nsec.

The operation of the two structures can be explained as follows:

If the it_interval field is empty(both sec and nsec) and the it_value consists of on Non-Zero values then it acts like a one-shot timer. If the it_interval field is not zero then the value is reloaded to it_value after every expiration.

The flow of the program I created :

1. The program entry point here “release_the_hounds” first deletes any currently running tasks if they have the name task_a and task_b, Then creates the two tasks with the corresponding priorities.
2. The tasks create timer A and timer B respectively with the proper resolution of 10 secs Below is a screen of the same



3. The handlers for the two tasks keep count for the number of executions and then cause the termination of the timers by calling the shutdown function which cancels and deletes the timer.

2. Fi-bonacci code

The Fibonacci implementation was fairy simple , it utilizes a function as shown below:

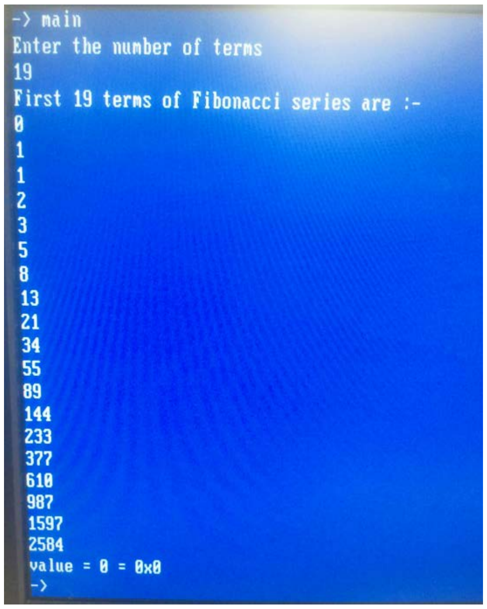
```
#include<stdio.h>
int fib()
{
   int n, first = 0, second = 1, next, c;
    printf("Enter the number of terms\n");
   scanf("%d",&n);
    printf("First %d terms of Fibonacci series are :-\n",n);
    for ( c = 0 ; c < n ; c++ )
   {
      if ( c <= 1 )
         next = c;
      else
      {
         next = first + second;
         first = second;
         second = next;
      }
      printf("%d\n",next);
   }
  return 0;
}
```

The initial implementation was with the print statements just to verify that the fi-bonaccci series was being generated up to the desired number.

The later implementation spawned the above function as a task and then I began inputting varying lengths of numbers as inputs to the function and observing the system viewer trace. The system viewer trace to determine what number of fi-bo generation can be sutable for 10ms and my approach was to double it for 20ms and ovserve the interval generated.

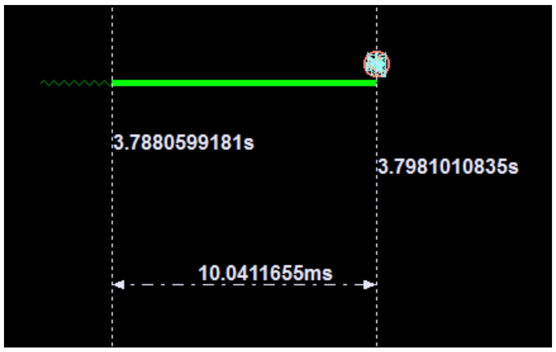
By a very iterative process the counts were found too be

n= 2431900; for 10ms   and n= 4863800; for 20ms

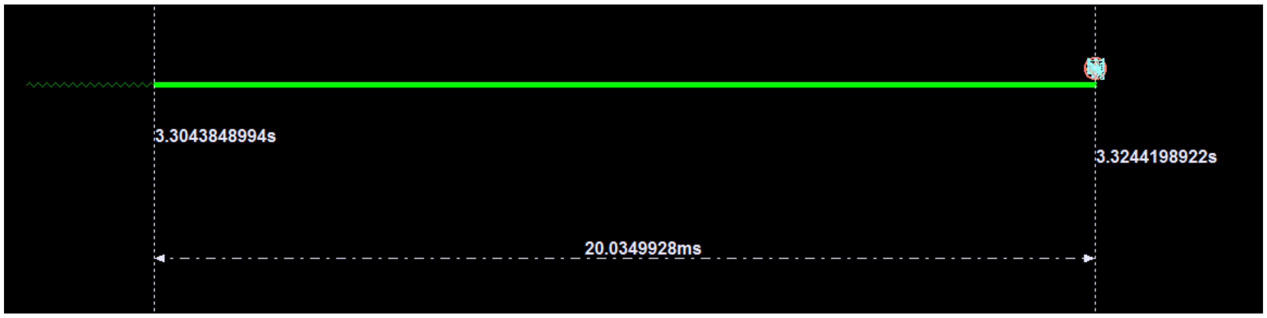


```
-> main
Enter the number of terms
19
First 19 terms of Fibonacci series are :-
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
value = 0 = 0x0
->
```

Fib_test for 10 ms



Fib_test for 20ms



3. Final Code for the implementation of the combination

This code is similar to the first implementations to perform the , The code is designed sot that you can change the values of T1 , T2,C1 and C2 right from the top of the code.

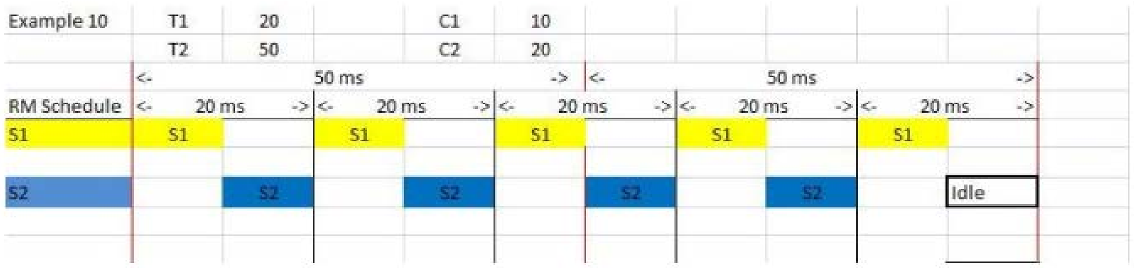
The flow the code is as follows:

1. The entry point function for the file is the “release_the_hounds” function and the system clock is set to 1000 so that the resolution is 1ms.
2. Here the timer parameters are set for their creation and the intervals along with the signal names are assigned.
3. The two task hound1 and hound2 are spawned with assigned priorities.
4. The tasks perform three important timer related functions,
    a. sigaction assigns the pairing of the event and the signal and then the time
    b. Timercreate() assignes the required parametes
    c. Timer_settime arms the timer

After this the pause function with the Fibonacci function is called within a forever while loop. This keeps the timer in the context of the task and does not allow the process thred to exit and allows the task to analyse the timer each time.

Belwo is a Screenshot of the

As for the LSD it would satisfy as the theorem states that if the tasks is found to be schedulable within the period wich forms the LCM of the two tasks. Then the task is found to be completely schedulabe as shown in the below diagram.



Example 10	T1	20		C1	10						
	T2	50		C2	20						
	<-		50 ms		->	<-		50 ms		->	
RM Schedule	<- 20 ms	->	<- 20 ms	->	<- 20 ms	->	<- 20 ms	->	<- 20 ms	->	
S1	S1		S1		S1		S1		S1		
S2		S2		S2		S2		S2		Idle	

Fig 1: Scheduling of tasks over LCM of periods (100 ms) when system is feasible.